UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Practical Byzantine Fault Tolerance, un algoritme de consens capaç de competir amb Proof-of-Work

Treball de Final de Grau

Presentat a la facultat de la

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

per

Sergi Ventura Roig

En compliment parcial dels requisits del grau en

ENGINYERIA TELEMÀTICA

Assessor UPC: Jose Luis Muñoz Tapia

Barcelona, Maig 2019

Abstract

This document shows the research on the Practical Byzantine Fault Tolerance consensus algorithm, the base that uses other protocols to achieve agreement on different values on a blockchain.

As introduction, a study of the Zookeeper Atomic Broadcast protocol has been done to introduce concepts and mechanics that will be used during the project.

The operation of the Practical Byzantine Fault Tolerance algorithm is analyzed, which has certain limitations when it is implemented in large systems, that is, in a real environment. But in recent years, this protocol has been used as the base to develop other protocols such as Federated Byzantine Agreement, a protocol based on PBFT that overcomes these limitations and allows the emergence of a very high-ranked cryptocurrencies on the sector.

We will explain that a considerable part of the current blockchain infrastructure could be replaced by protocols such as the Federated Byzantine Agreement that are consensus algorithms based on Practical Byzantine Fault Tolerance. We also analyze the advantages that these protocols have, regarding other technologies such as Proof-of-work, which is used by Bitcoin, the digital currency with the highest market value at this moment.

Finally, slides have been generated for the Zookeeper Atomic Broadcast protocol, the Practical Byzantine Fault Tolerance and the Federated Byzantine Agreement. A python simulation of the Zookeeper Atomic Broadcast algorithm has also been programmed to observe different system behaviors depending on the state of its nodes.

Resum

Aquest document mostra la investigació sobre l'algoritme de consens *Practical Byzantine Fault Tolerance*, la base que utilitzen altres protocols per aconseguir acord sobre diferents valors a una *blockchain*.

Com a introducció s'ha fet un estudi del protocol *Zookeeper Atomic Broadcast*, per introduir conceptes i mecàniques que s'utilitzaran durant el treball.

S'analitza el funcionament de l'algoritme *Practical Byzantine Fault Tolerance*, el qual, té certes limitacions a l'hora de posar-se en pràctica en sistemes molt grans, és a dir, en un entorn real. Però en els darrers anys s'ha utilitzat aquesta base per desenvolupar protocols com és el cas de *Federated Byzantine Agreement*, un protocol basat en PBFT que es sobreposa a aquestes limitacions i permet el sorgiment de criptomonedes amb molt pes en l'actual mercat de les *blockchain*.

Donarem arguments per defensar que una bona part de la infraestructura actual en l'àmbit de les *blockchain* es podria substituir per protocols com *Federated Byzantine Agreement*, és a dir, algoritmes de consens basats en *Practical Byzantine Fault Tolerance*. També s'analitzen les avantatges que ens donen aquests protocols respecte altres tecnologies com *Proof-of-work*, la qual és utilitzada per Bitcoin, l'actual moneda digital de major valor al mercat.

Per acabar, s'ha generat material docent en forma de diapositives dels protocols *Zookeeper Atomic Broadcast, Practical Byzantine Fault Tolerance* i *Federated Byzantine Agreement.* També s'ha realitzat una simulació en *python* de l'algoritme *Zookeeper Atomic Broadcast* amb la finalitat d'observar diferents comportaments del sistema depenent de l'estat dels nodes que el formen.

Resumen

Este documento muestra la investigación sobre el algoritmo de consenso *Practical Byzantine Fault Tolerance*, la base que utilizan otros protocolos para lograr el acuerdo sobre diferentes valores en una *blockchain*.

Como introducción se ha hecho un estudio del protocolo *Zookeeper Atomic Broadcast*, para introducir conceptos y mecánicas que se utilizarán durante el trabajo.

Se analiza el funcionamiento del algoritmo *Practical Byzantine Fault Tolerance*, el cual, tiene ciertas limitaciones a la hora de ponerse en práctica en sistemas muy grandes, es decir, en un entorno real. Pero en los últimos años se ha utilizado esta base para desarrollar protocolos como es el caso de *Federated Byzantine Agreement*, un protocolo basado en PBFT que se sobrepone a estas limitaciones y permite el surgimiento de criptomonedas con mucho peso en el actual mercado de las *blockchain*.

Daremos argumentos para defender que una buena parte de la infraestructura actual en el ámbito de las *blockchain* se podría sustituir por protocolos como *Federated Byzantine Agreement*, es decir, algoritmos de consenso basados en *Practical Byzantine Fault Tolerance*. También se analizan las ventajas que nos dan estos protocolos respecto otras tecnologías como *Proof-of-work*, la cual es utilizada por Bitcoin, la actual moneda digital de mayor valor en el mercado.

Por último, se ha generado material docente en forma de diapositivas de los protocolos *Zookeeper Atomic Broadcast*, *Practical Byzantine Fault Tolerance* y *Federated Byzantine Agreement*. También se ha realizado una simulación en *python* del algoritmo *Zookeeper Atomic Broadcast* con el fin de observar diferentes comportamientos del sistema dependiendo del estado de sus nodos.

Dedicatòria

M’agradaria dedicar aquest treball a la meva família, per permetre que hagi estudiat aquesta carrera i el suport que m’han donat. Als companys de grau, doncs han sigut de molta ajuda en els moments més complicats durant aquesta etapa. Als professors per generar-me interès en el camp de les telecomunicacions. I per últim al José Luis Muñoz per descobrir-me el món de les blockchain i dels algoritmes de consens.

Historial de revisions y registre d'aprovació

Revisió	Data	Propòsit
0	12/03/2019	Creació del document
1	10/04/2019	Revisió del document
2	25/04/2019	Revisió del document
3	5/05/2019	Revisió del document
4	8/05/2019	Revisió del document
5	10/05/2019	Revisió final del document

LLISTA DE DISTRIBUCIÓ DEL DOCUMENT

Nom	Mail
Sergi Ventura Roig	sergi.ventura@hotmail.com
Jose Luis Muñoz Tapia	jose.munoz@entel.upc.edu

Escrit per:		Revisat i aprovat per:	
Data	8/05/2019	Data	10/05/2019
Nom	Sergi Ventura Roig	Nom	Jose Luis Muñoz Tapia
Posició	Autor del projecte	Posició	Supervisor del projecte

Taula de continguts

Llista de figures

1. Introducció

Abans d'explicar els objectius de l'estudi o la hipòtesi d'aquest és necessari explicar què és un algoritme de consens i perquè s'utilitza.

Un algoritme de consens es un procés informàtic que té com a objectiu arribar a un acord sobre certs valors en un sistema distribuït, on pot incloure múltiples nodes poc fiables. Arribar a un consens és fàcil en absència de fallades però es torna en un objectiu més complex en presencia de comportaments erronis, aleatoris o maliciosos. Una de les característiques d'aquests algoritmes es que sempre tindran un màxim d'errors que podran suportar però si es supera aquest número, en cap cas es podrà garantir el correcte funcionament del protocol. En aquest document s'explica com varia aquesta cota depenent del tipus d'algoritme que s'utilitza.

Pel que fa a les criptomonedes o monedes digitals, els algoritmes han d'assegurar que les transaccions siguin vàlides i que la confirmació arribi a una gran quantitat de participants. També han d'evitar accions que posin en perill la blockchain com el doble pagament o el trencament d'un smart contract sense complir amb els requisits necessaris.

En l'actualitat, i des del naixement de les blockchain, l'algoritme de consens més utilitzat és Proof-of-Work, conegut per les sigles POW. Aquest és utilitzat per moltes monedes digitals, entre elles Bitcoin.

1.1. Objectius

L'objectiu principal d'aquest treball des d'un principi era estudiar diferents algoritmes de consens i decidir en quines situacions s'havia d'utilitzar cada un d'ells. Més tard es va veure que molts d'aquests eren similars i es va decidir centrar l'estudi en un algoritme de consens que funcionés en un àmbit fail-stop, com és el cas del protocol Zookeper Atomic Broadcast i en un algoritme capaç de suportar atacs al sistema com és el cas del protocol Practical Byzantine Fault Tolerance, una alternativa a PoW.

A l'analitzar els seus punts forts i febles per saber en quines situacions es podria implementar millorant el rendiment de PoW va sorgir la nostra hipòtesi inicial: Es pot reemplaçar gran part de la infraestructura actual basada en PoW per tecnologia PBFT. Es va decidir que l'anàlisi del protocol Zookeeper Atomic Broadcast també tindria un pes

important en aquest estudi doncs al comparar els dos protocols es pot veure la gran diferència que hi ha entre els algoritmes que poden sobreposar-se a atacs o els que no.

Un dels altres objectius que s'han assolit és la creació de material docent en forma de diapositives pels diferents protocols estudiats i una simulació en python de l'algoritme *Zookeeper Atomic Broadcast* per poder observar diferents comportaments del sistema

1.2. Sistemes distribuïts

Un sistema distribuït és un model en el qual els components de la xarxa es comuniquen entre si i coordinen les seves accions. Els sistemes que s'han estudiat són sistemes distribuïts que necessiten un algoritme de consens per arribar a un acord sobre algun valor. Per exemple fer *commit* d'una transacció de moneda digital en particular (*commit* és una paraula que s'agafa de l'anglès que significa que hi ha un compromís sobre un valor entre els diferents participants del sistema.

En aquests sistemes no hi ha memòria compartida, sinó que cadascun dels elements té la seva memòria local i és a on s'apliquen els algoritmes de consens per poder garantir el seu correcte funcionament. Aquests sistemes permeten més escalabilitat i són més robustos que sistemes centralitzats, on una fallada a l'entitat primària acabaria amb el correcte funcionament d'aquest.

S'ha d'introduir la figura de client, la qual el seu propòsit és introduir una transacció a la xarxa a través d'un node del sistema. Per aconseguir que aquests sistemes distribuïts es vagin actualitzant a mesura que els clients introdueixin noves sol·licituds, es necessita replicar l'estat dels diferents nodes. En els últims anys s'han utilitzat principalment dos tipus d'algoritmes de consens per realitzar-ho, de replicació passiva i de replicació activa.

1.2.1. Replicació passiva

En aquest tipus de replicació, les sol·licituds de cada client són processades per un sol servidor, anomenat primari. Quan aquest ha processat la sol·licitud, envia el nou estat actualitzat als altres nodes, anomenats *followers* o seguidors, i envia una resposta al client. En el suposat cas de que el servidor primari falli, un dels node *follower* ocuparà el seu lloc.

En el capítol 2 es veurà com aquest canvi de node primari s'haurà de fer de la manera més eficient possible per perdre el mínim de informació i el més ràpid possible doncs durant el procés es pararà el sistema, i per tant, es retardaran les respostes.

El principal desavantatge de la replicació passiva es que tot i ser un sistema distribuït, segueix sent molt centralitzat al tenir totes les competències sobre el node primari. Els algoritmes de replicació passiva també s'anomenen algoritmes de replicació d'estat.

1.2.2. Replicació activa

Les sol·licituds de cada client no són només processades per un servidor sinó per tots els servidors. Per aconseguir que aquests acabin en el mateix estat es necessita que el procés sigui determinista, és a dir, donat el mateix estat inicial i una sol·licitud, tots els nodes produiran la mateixa resposta i acabaran en el mateix estat final. També es necessita que aquests rebin els missatges en el mateix ordre ja que sinó podrien finalitzar en estats diferents. Per exemple un individu que gastés una quantitat de moneda digital abans de que la cobrés. D'aquesta manera, es detectaran les falles de manera simple quan l'estat o la sortida d'algun node variïn respecte al de la majoria.

El gran avantatge de la replicació activa es que permet la descentralització al no tenir un node que actuï com a principal. En aquest tipus de replicació els servidor s'anomenen rèpliques. Els algoritmes de replicació activa també s'anomenen algoritmes de replicació de màquines d'estat.

1.2.3. Requisits

Tot algoritme de consens ha de complir amb dos requisits per ser útil. Aquests s'aniran citant durant el cos del treball per veure en quines situacions un algoritme pot garantir o no el correcte funcionament d'un sistema.

- *Safety*: Dues rèpliques no poden decidir dos valors diferents per una mateixa variable, resultat o transacció i aquest valor ha de ser proposat per alguna rèplica. És a dir, no pot ser imposat per un client.

- *Liveness*: Totes les rèpliques fiables han d'acabar escollint algun valor eventualment.

1.3. Estructura del document

La resta del document està estructurat com es detalla a continuació:

Capítol 2: S'introdueix el protocol Zookeeper Atomic Broadcast, un algoritme de consens que utilitza el model fail-stop.

Capítol 3: S'analitza el protocol Practical Byzantine Fault Tolerance i es mencionen les seves principals limitacions.

Capítol 4: Es compara PBFT amb Proof-of-Work i Proof-of-Stake, les seves principals competidores.

Capítol 5: Es descriu el protocol Federated Byzantine Agreement. S'expliquen les millores respecte PBFT i punts forts respecte les principals competidores

Capítol 6: Menció de les criptomonedes que utilitzen algun tipus de tecnologia derivada de PBFT i valor al mercat d'aquestes

Capítol 7: Conclusions i reflexions sobre la hipòtesi inicial

2. Zookeeper Atomic Broadcast Protocol

2.1. Introducció a l'algoritme

Zookeeper Atomic Broadcast Protocol o més conegut amb les sigles ZAB és el protocol utilitzat per Zookeeper. És un algoritme de replicació passiva i funciona amb el model fail-stop. Aquest tipus d'algoritmes fail-stop s'utilitzen en escenaris controlats on no s'espera que siguin atacats sinó que l'únic que pot passar és que un node deixi de funcionar, ja sigui per algun problema intern o simplement per una desconnexió. Alguns exemples d' aplicacions que utilitzen algoritmes d'aquest tipus són emmagatzematge de dades de configuració o arrencades inicials de sistemes entre d'altres.

Intuïtivament, es pot deduir que es necessita com a mínim tres nodes perquè un sistema fail-stop pugui treballar correctament, on un node podrà patir una desconnexió i els altres dos compararan l'estat i la sortida entre ells per arribar a una conclusió sobre la transacció introduïda. Si coincideixen sabran que ells funcionen correctament i el tercer node no. En el suposat cas de tenir només dos nodes i una desconnexió, no els hi serà possible fer aquesta verificació.

En general:

$n \geq 2f + 1$; on n és el número de nodes del sistema i f el número de fallades que el sistema pot suportar

Per aconseguir que s'arribi sempre a un acord, ZAB li dóna molta importància a l'ordre en el que s'envien i processen les transaccions. Exactament Zab busca complir amb aquests objectius:

- Entrega fiable: Si es fa *commit* d'una transacció a 1 servidor, es farà *commit* de la transacció a tots els servidors

- Ordre total: Si es fa *commit* d'una transacció A abans que una transacció B a un servidor, es farà commit de la transacció A abans que de B a tots els servidors.

- Ordre causal: Si una transacció A s'ha enviat abans que una transacció B, es farà *commit* de la transacció A abans que de B a tots els servidors. Per poder garantir-ho es necessita que els missatges no es desordenin quan s'envien per la xarxa. Per tant, ZAB utilitza canals *FIFO* per l'intercanvi de missatges per evitar aquests desordres.

Definició: ZooKeeper transaction id o zxid és un número de 64 bits que permet ordenar les diferents transaccions del sistema per tal de complir amb els requeriments d'ordre total i causal. Zxid es divideix en dos números de 32 bits:

- El número de 32 bits de més pes indica el líder i s'anomena epoch.

- El número de 32 bits de menys pes indica la transacció i s'anomena counter.

2.2. Fases de l'algoritme

En aquest apartat s'expliquen les diferents fases per les que passa el sistema fins que aquest està sincronitzat, és a dir, el moment en el que la majoria dels nodes tenen el mateix registre i quan un client entri una transacció, aquesta sigui commitejada.

2.2.1. Elecció de líder

ZAB és un protocol de replicació passiva i per tant hi ha un servidor que actua com a líder o també anomenat node primari. Per defecte s'utilitza l'algoritme Fast Leader Election, en el qual, surt escollit com a node primari aquell amb el zxid més gran. Això és, el node amb l'estat més actualitzat. Tots els nodes disposen d'un identificador i en el cas de tenir més d'un amb l'estat més actualitzat, el primari serà aquell amb el identificador més baix. (generalment el que va entrar al sistema abans)

Tanmateix, no és necessari utilitzar aquest algoritme doncs es pot utilitzar qualsevol altre tipus de protocol d'elecció de líder sempre i quan compleixi amb les següents condicions:

- Acabament: Un cop el líder ha sigut escollit l'algoritme ha d'acabar per evitar un possible segon canvi de líder provocant un retard innecessari

- Unicitat: Existeix exactament un node que és considera a ell mateix com a líder

- Acord: Tots els nodes del sistema saben qui és el líder

En el següent apartat es veu com en cap cas un algoritme diferent a Fast Leader Election serà més eficient que aquest.

2.2.2. Establiment

Aquesta fase dependrà molt de si s'ha utilitzat el protocol Fast Leader Election o no. En el cas de que no, si el primari no és el node amb l'estat més actualitzat, aquest s'ha d'actualitzar a partir de les transaccions guardades als altres nodes fins que aquest tingui la última transacció amb el zxid més alt. En el cas de que s'hagi utilitzat el protocol Fast

Leader Election per escollir el node primari, aquest pas no serà necessari doncs aquest ja és el servidor més actualitzat.

En aquest moment, cada node seguidor li envia al primari el zxid de la última transacció que tenen guardada amb l'objectiu de que aquest els enviï totes les transaccions que els hi resten amb el zxid més alt. (Aquest procés ja forma part de la fase de sincronització). El node primari estableix un nou epoch per evitar propostes de l'anterior primari i a partir d'aquest moment totes les noves transaccions tindran el zxid de la forma:

Zxid = ($e_{anterior}$+1, counter) ; on el counter començarà a 0 i s'anirà augmentant seqüencialment

2.2.3. Sincronització

Aquesta fase comença quan el primari sap l'estat dels seguidors i comença a proposar transaccions del seu historial amb el nou epoch amb l'objectiu d'actualitzar l'estat de tots. Al final d'aquesta fase tots els nodes del sistema que funcionin correctament de la xarxa han de tenir el mateix estat.

En el moment en que un node seguidor rep una proposta del líder, li enviarà un acknowledge. Cal recordar que al ser un protocol fail-stop es necessita que més de la meitat dels nodes funcionin correctament, per tant, el líder necessitarà f acks per poder fer *commit* d'una transacció. Es possible que no hi hagin prou nodes funcionant correctament per realitzar-ho si hi ha més nodes caiguts dels f nodes que la xarxa pot suportar però eventualment algun node es reconnectara. (No són f+1 acks ja que el node primari no n'envia). Aquest intercanvi de missatges es pot observar més detalladament a la figura 2.1.

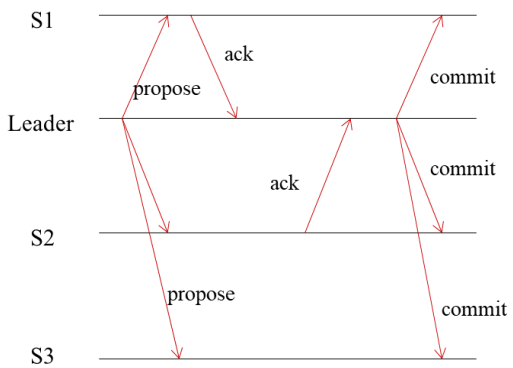

Figura 2.1 Missatgeria intercanviada entre el líder i els seguidors a les fases de sincronització i broadcast

2.2.4. Broadcast

Quan tots els nodes estan sincronitzats comença la fase de broadcast i dura indefinidament si no es detecta cap error que provoqui la necessitat d'un canvi de líder. Quan algun client entra una transacció al sistema, el node primari fa un broadcast d'aquesta als seguidors i com a la fase de sincronització, si rep f acks farà commit de la transacció. L'esquema d'intercanvi de missatges és el mateix que la de la fase de sincronització i la de la figura 2.1.

2.2.5. Recuperació

Com s'ha comentat anteriorment el protocol funciona correctament fins que el primari falla. En general, sempre hi haurà almenys un servidor que tindrà registrades totes les transaccions que han sigut proposades per l'anterior primari i aquest serà escollit com al nou node primari. En cap cas se li permet al nou primari reordenar transaccions. Haurà de seguir l'ordre proposat per l'anterior líder i dels clients per complir amb els requeriments anteriorment mencionats.

En el cas de que el líder es desconnecti de la xarxa amb transaccions pendents per enviar, s'escull com a primari el node amb l'estat més actualitzat per tal de perdre el menor número de transaccions possibles. Quan un client introdueix una transacció al sistema es posa en marxa un temporitzador. Si el client no rep cap notificació del node primari passat el temps màxim del temporitzador, reenviarà la transacció. Aquest procediment s'utilitza per evitar aquestes possibles pèrdues degut a la fallada del primari.

2.3. Escenaris

En aquest apartat s'analitza un sistema de 5 nodes des d'un estat inicial on la xarxa funciona correctament fins a desconnexions de nodes varies per tal d'entendre com es comporta el protocol en diferents escenaris. Un sistema amb 5 nodes pot suportar 2 fallades ja que:

$n \geq 2f + 1 \quad ; \quad 5 \geq 2 \cdot 2 + 1 = 5$

2.3.1. Escenari 1

En el primer escenari com es pot observar a la figura 2.2, tots els nodes funcionen correctament i comencen des de l'estat inicial Zxid = [0,0]. S'escull al node 1 com a primari doncs en el cas de tenir el mateix zxid, nosaltres escollim el node amb

l'identificador més petit. En aquest cas és indiferent quin node s'escull ja que tots tenen el mateix estat. El client envia dues transaccions.

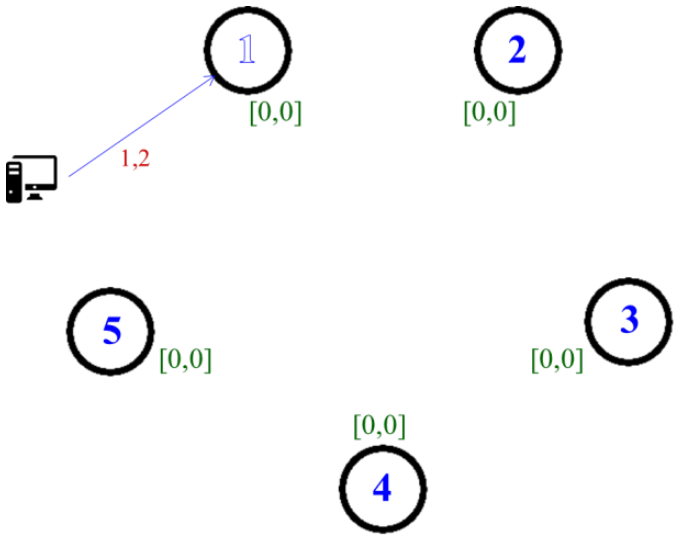

Figura 2.2 Estat inicial del sistema de l'escenari 1

Al no tenir cap desconnexió de la xarxa, totes les transaccions es processen als nodes. Com que el líder rep f acks o més, fa *commit* de les dues transaccions i ho comunica al client. A la figura 2.3 es pot observar l'estat final del sistema.

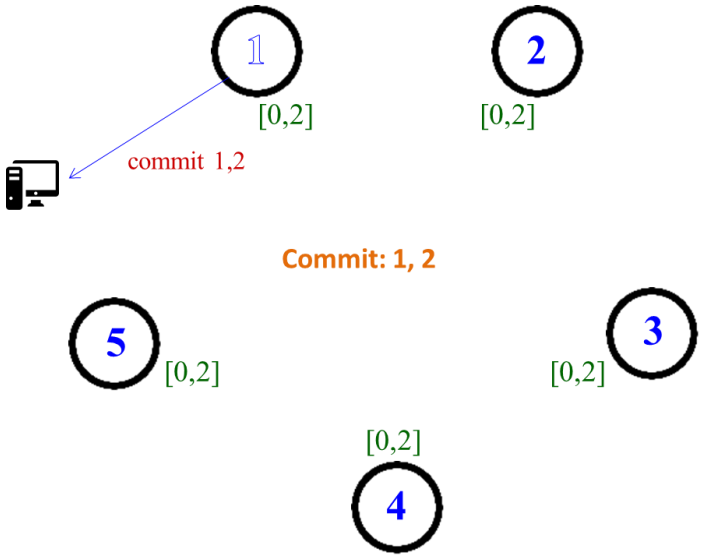

Figura 2.3 Estat final del sistema de l'escenari 1

2.3.2. Escenari 2

En aquest cas, els nodes 4 i 5 s'han desconnectat de la xarxa com es pot observar de color vermell a la figura 2.4. S'envien dues transaccions i el node 3 es desconnecta just després de processar la primera transacció

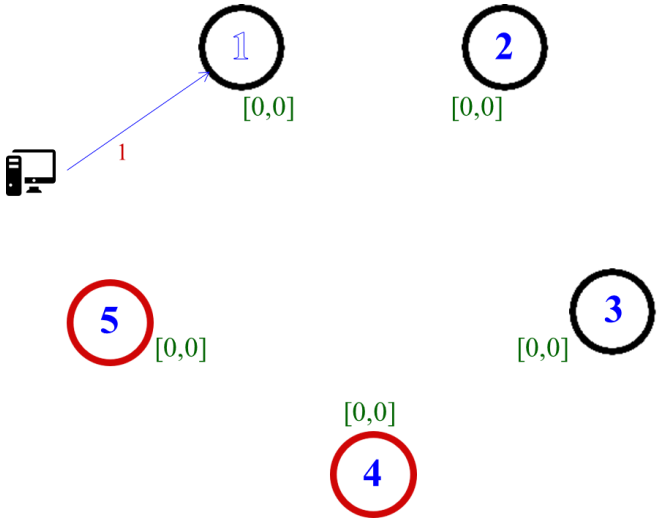

Figura 2.4 Estat inicial del Sistema de l'escenari 2

El primari podrà fer *commit* de la primera transacció doncs rebrà 2 acks (dels nodes 2 i 3) però no podrà fer *commit* de la segona doncs només en rebrà un. Per fer-ho, el node primari haurà d'esperar a que es reconnecti algun altre node seguidor. L'estat final del sistema es pot observar a la figura 2.5.

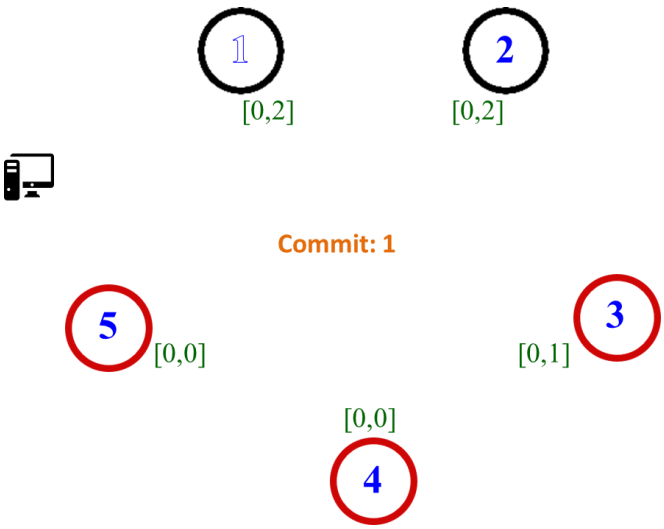

Figura 2.5 Estat del sistema de l'escenari 2 després de fer commit de la 1a transacció

2.3.3. Escenari 3

Sortint de l'estat anterior, els nodes 3,4,5 es reconnecten correctament a la xarxa però el primari deixa de funcionar correctament com s'observa a la figura 2.6.

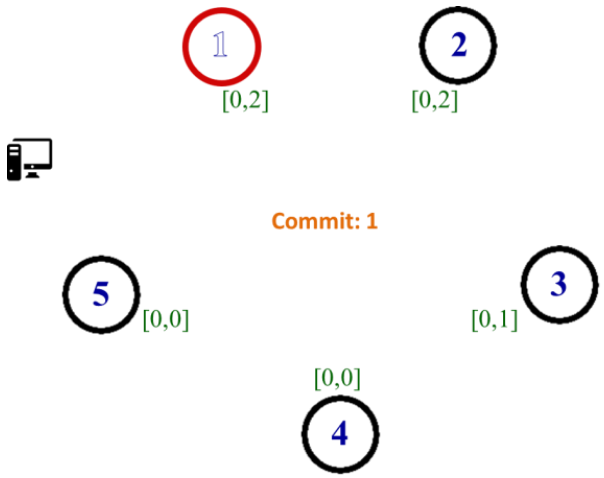

Figura 2.6 Estat inicial del sistema de l'escenari 3

Aleshores un quorum de mínim 3 nodes en aquest escenari, farà eleccions pel canvi de líder. En aquest escenari utilitzem el protocol Fast Leader Election per escollir el node primari i per tant serà el que tingui grabada la transacció amb el zxid més alt. En aquest escenari el nou primari serà el node 2 doncs és l'únic servidor funcionant correctament que té processada la transacció [0,2].

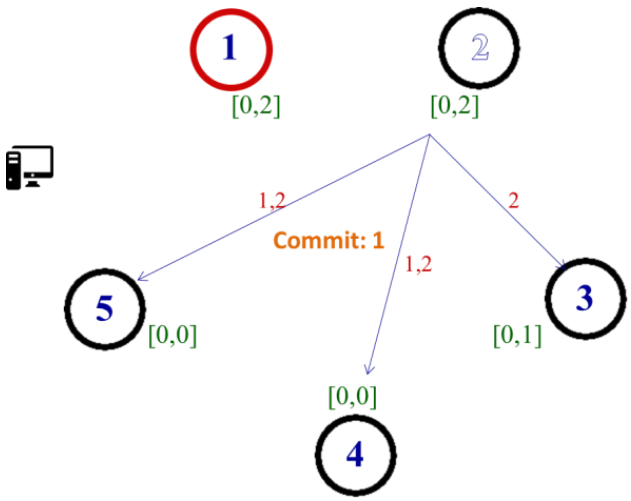

Figura 2.7 Estat del sistema de l'escenari 3 després d'escollir el nou primari

En aquest moment, el node 2 estableix un nou epoch i proposa la transacció 1 als nodes 4 i 5 i la transacció 2 als nodes 3,4,5 com es pot observar a la figura 2.7. Com que aquests 3 nodes funcionen correctament, enviaran els corresponents acks i es podrà fer *commit* de la segona transacció. És important mencionar que el node 2 no cal que enviï la transacció 1 al node 3 amb el nou epoch, doncs aquest ja la té registrada amb l'epoch anterior.

3. Practical Byzantine Fault Tolerance

3.1 Introducció

En el capitol 2 s'ha analitzat a profunditat un algoritme de consens de model fail-stop. Amb un model d'aquest tipus s'han introduit conceptes i mecanismes que utilitzen les blockchains però com s'ha esmentat, només són utilitzats en certes aplicacions. En aquest capítol introduim els errors bizantins que ens permetran explicar acuradament com funcionen les blockchains.

Un error Bizantí es aquell provocat per un node defectuós degut a un error de software que pot sorgir d'un error elèctric o degut a un atac maliciós. Aquest pot provocar un comportament arbitrari com un error al tornar un resultat o directament un resultat erroni. Pot passar també que un node maliciós generi dades arbitràries fingint ser un node funcionant correctament. Això implica que els errors Bizantins poden confondre els sistemes de detecció de fallades, cosa que fa difícil la tolerància a aquestes.

En un àmbit de moneda virtual, els atacants tenen un gran incentiu de cara a entrar maliciosament a la xarxa. Un exemple d'atac seria un individu intentant gastar alguna quantitat de moneda virtual sense ser el propietari d'aquesta per exemple o sortir d'un contracte virtual entre participants del sistema on cada un d'ells té unes obligacions, aquests contractes són coneguts amb el terme *smart contract*.

Per evitar aquestes intrusions a una blockchain es necessita un algoritme de consens capaç d'impedir la falsificació d'entrades i que d'alguna manera permeti avaluar les sortides per tal de mai fer *commit* d'una transacció provinent d'un node maliciós.

3.1.1. Problema del general bizantí

Per entendre d'on sorgeix i quin és l'objectiu del protocol PBFT és necessari explicar el problema del general bizantí. Imaginem que unes quantes divisions d'un exercit bizantí estan rodejant una ciutat enemiga esperant per atacar. Cada divisió està comandada per un general i aquests es comuniquen entre ells a través de missatgers.

Després d'observar a l'enemic han de decidir si ataquen o no. Però si ataquen, necessiten tenir una forta majoria per poden guanyar la batalla, sabent que alguns dels generals poden ser traïdors i que intentaran evitar que els lleials arribin a un acord.

En definitiva, els generals necessiten un algoritme que garanteixi:

- Un petit nombre de traïdors no pot alterar la decisió dels lleials

- Tots els generals lleials decideixen el mateix pla d'acció independentment del que facin els traïdors, o ataquen tots o es queden tots a l'espera.

En el àmbit de les blockchain també es necessita complir amb els mateixos requisits doncs s'ha de fer *commit* d'una transacció sempre i quan es tingui el suficient número de nodes lleials per garantir que els intercanvis de missatges no han sigut atacats.

3.1.2. Introducció a l'algoritme

PBFT es un protocol de replicació activa que tolera els errors bizantins en un xarxa on els nodes estan connectats entre si. Recordem que en aquests tipus de protocols els nodes s'anomenen rèpliques. L'algoritme està dissenyat per funcionar en sistemes asíncrons amb un temps d'execució molt baix.

Totes les rèpliques comencen en el mateix estat i són deterministes, és a dir, l'execució d'una operació, donat un estat i uns arguments, sempre produeix el mateix resultat. Les rèpliques es comuniquen entre si i l'objectiu és que totes les lleials arribin a un acord sobre l'estat del sistema. Quan una rèplica rep una petició del client, executa l'operació d'aquesta i decideix per ella mateixa si el missatge es fiable o no. Aleshores comparteix aquesta decisió amb les altres.

Tots els missatges que s'intercanvien en el sistema han de demostrar que venen d'un node específic i han de verificar que no han sigut modificats durant la transmissió. Per realitzar-ho, tots els missatges que s'envien porten una signatura de clau pública i el *digest* del missatge produït per funcions de hash resistents a col·lisions. Totes les repliques coneixen les claus publiques de les altres per poder verificar les signatures.

Perquè l'algoritme funcioni, s'assumeix que un atacant no pot retardar el sistema indefinidament i no pot subvertir les tècniques criptogràfiques esmentades. Per exemple, produir una signatura vàlida d'un node no defectuós, calcular la informació d'un *digest* o trobar dos missatges amb el mateix *digest*.

El protocol garanteix *safety* i *liveness* sempre que la quantitat de nodes maliciosos de la xarxa sigui inferior a 1/3 dels nodes del sistema.

Demostració:

Per poder validar qualsevol transacció, es necessita que el número de nodes que la verifiquin sigui superior al número de nodes funcionant incorrectament. En el cas de ZAB, com s'ha vist al capítol 2, tenim:

$n-f > f \rightarrow n > 2f$

Però a l'introduir els errors bizantins a aquesta fórmula, pot passar que es rebin f sortides generades per nodes maliciosos fent creure que són sortides provinents de nodes lleials. De manera que en aquestes n sortides que es reben pot haver-hi f sortides malicioses. Per tant es necessita una seguretat extra de f sortides per assegurar majoria en el cas de que tots els nodes maliciosos enviessin una sortida errònia. Així doncs:

$(n-f)-f > f \rightarrow n > 3f$; on n és l número de nodes del sistema i f el número de nodes maliciosos

Cal mencionar també el concepte de client maliciós, que seria aquell que té com a objectiu invocar alguna operació encara no acceptada a la xarxa. En aquest cas el protocol assegura que aquests accessos son observats constantment per tots els altres clients proporcionant una potent defensa en contra dels atacs dels clients maliciosos. Per tant, *safety* es proporciona independentment de quants d'aquests clients utilitzin el servei, ja que no podran alterar el funcionament del sistema

3.2. Protocol PBFT

Les rèpliques del sistema es mouen a través d'una successió de configuracions anomenades vistes. En cada una d'elles, una rèplica fa la funció de líder o primari mentre que les altres actuen com a nodes *followers*. Les vistes es numeren consecutivament i es mouen a la següent quan el primari falla. El primari d'una vista és:

$P = v \bmod |R|$, on v és el número de vista i |R| el nombre de nodes del sistema

Passen 4 etapes fins a aconseguir fer *commit* d'una transacció:

1- Un client envia una petició o *request* al node líder amb la transacció.

2- El líder fa un multicast del *request* cap als altres nodes.

3- Els nodes followers executen l'operació vinculada al request i envien un missatge de resposta o *reply* al client.

4- El client espera f+1 respostes dels diferents nodes amb el mateix resultat.

Durant l'explicació de les diferents fases es mencionen els missatges enviats entre nodes amb el format $[X,a,b,c]\sigma_s$ on X és el tipus de missatge, a b c els arguments i σ_o la signatura de clau pública del node on s'origina el missatge

3.2.1 Etapes principals de l'algoritme

El protocol comença quan un client envia un *request* amb la corresponent operació (transacció) al node que ell creu que és el primari. El client a priori coneix la identitat del primari doncs tots els missatges que s'envien tenen com a paràmetre el número de vista. El missatge request té com a paràmetres l'operació (o) que es vol aplicar a la xarxa, un registre d'hora anomenat *timestamp (t)* per poder ordenar els diferents requests de tots els clients, el seu identificador (c), i com tots els missatges, la seva signatura de clau pública (σ_c).

El missatge $[REQUEST,o,t,c]\ \sigma_c$ s'envia com es mostra a la figura 3.1

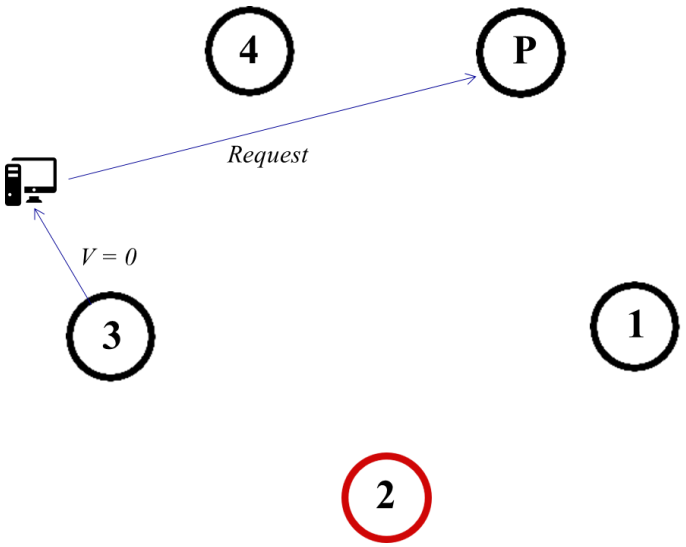

Figura 3.1 Missatge de request del protocol PBFT

En el cas de que el missatge sigui enviat cap a un node *follower*, degut a un possible canvi de primari sense que el client s'assabentés, el request seria redirigit pel d'aquest cap al primari.

En el moment en que el request arriba al node primari comença una etapa de 3 fases perquè el request arribi als altres nodes amb tot tipus de seguretat per garantir l'execució de l'operació de manera correcte i fiable. Les 3 fases són anomenades pre-prepare, prepare i commit. Les dues primeres serveixen per ordenar el request en una mateixa

vista. Les fases de prepare i commit també s'encarreguen que els request que s'han processat correctament estiguin ordenats entre diferents vistes.

A la fase del pre-prepare, el primari assigna un número de seqüència al request i fa un multicast d'aquest i del missatge pre-prepare. Aquest té com a paràmetres el número de vista (v), el número de seqüència (n), el *digest* del missatge (d) i la seva signatura de clau pública (σ_p). La finalitat d'enviar aquest missatge es només aquesta assignació i per tant no s'inclou el request sencer, sinó el seu *digest*. D'aquesta manera s'obté un paquet més petit i es pot utilitzar un mètode de transport més eficient per paquets més petits i un altre per paquets més grans i així reduir el temps d'execució.

Els missatges [PRE-PREPARE,v,n,d] σ_p i el *request* s'envien com es mostra a la figura 3.2.

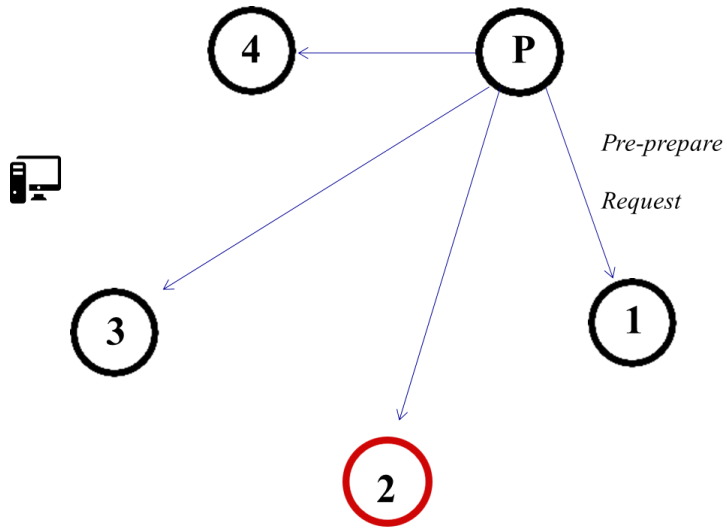

Figura 3.2 Missatge de pre-prepare del protocol PBFT

Definició: *Low water mark* (h) i *high water mark* (H) són dues cotes de l'espai dels números de seqüència, h és la cota inferior i H la superior. La seva finalitat es evitar que un node primari maliciós pugui acabar amb l'espai de números de seqüencia escollint un número molt alt. Més endavant s'explica com a mesura que es necessiten més números de seqüencia es va expandint aquest marge.

Un node *follower* accepta el missatge pre-prepare i entra a la fase prepare si:

- d és realment el *digest* de m.

- Les signatures del request i del missatge pre-prepare són correctes.

- v és el número de vista actual.

- No ha sigut acceptat cap missatge pre-prepare per la vista v i número de seqüencia n amb un *digest* diferent, per tal d'evitar que un possible atacant canviï el número de seqüència alterant el funcionament del protocol.

- n es troba entre els límits h i H (h <= n <= H).

Quan un node *follower* entra a la fase prepare, realitza un multicast del missatge de prepare cap a totes les altres rèpliques. Aquest missatge no inclou més paràmetres respecte al pre-prepare més que la identificació (i) del node *follower* que l'envia. És simplement un acknowledge perquè el primari i els altres nodes sàpiguen que s'ha assignat el número de seqüència n per la vista v pel request amb el *digest* d.

Els missatges de [PREPARE,v,n,d,i] σ_i s'envien com es mostra a la figura 3.3

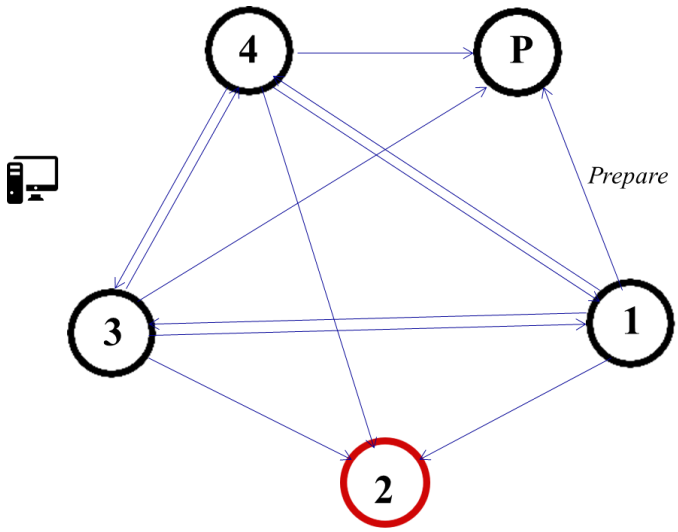

Figura 3.3 Missatge de prepare del protocol PBFT

Cada node guarda en el seu registre el request i els missatges de pre-prepare i prepare. Més endavant s'explica el mecanisme per anar netejant aquests registres quan ja s'ha fet *commit* de les transaccions vinculades a aquests missatges.

Qualsevol rèplica, incloent la primària, accepta els missatges prepare si es compleixen les mateixes condicions esmentades anteriorment a la fase de pre-prepare, i seran les condicions necessàries per acceptar qualsevol tipus de missatge.

Definició: Prepared(m,v,n,i) és cert si i només si la rèplica i ha introduït en el seu registre:

- el request m.

- un missatge pre-prepare per m a la vista v amb el número de seqüència n.

- 2f missatges de prepare incloent el seu propi que corresponguin al pre-prepare[1].

En aquest moment, els nodes concorden en que el request m està a la vista v amb el número de seqüència n.

Quan prepared(m,v,n,i) es cert, el node i entra a la fase de commit. Aquest fa un multicast del missatge de commit cap a totes les altres rèpliques. Els altres nodes accepten el missatge si es compleixen totes les condicions esmentades. Aquest missatge té els mateixos paràmetres que els missatges de pre-prepare i prepare.

Els missatges de [COMMIT,v,n,d,i] σ_i s'envien com es mostra a la figura 3.4.

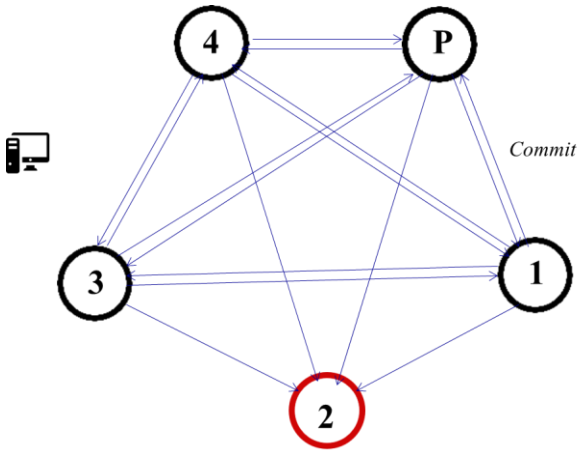

Figura 3.4 Missatge de commit del protocol PBFT

Definició: Commited-local(m,v,n,i) es cert si i només si prepared(m,v,n,i) es cert i la rèplica i ha acceptat 2f + 1 missatges de commit, incloent el seu, que corresponguin al pre-prepare. És a dir si tenen la mateixa vista, número de seqüència i *digest*.

Només que un request s'hagi "commitejat localment" a una rèplica fiable, eventualment ho farà a f+1 rèpliques ja que vol dir que ha acceptat 2f+1 missatges de commit i per tant, encara que hi hagi algun problema a la xarxa, aquests missatges acabaran arribant a les altres rèpliques lleials. Degut a aquests problemes, es possible que els *requests* es guardin als nodes desordenadament. Tot i així, no implica que l'algoritme no satisfaci els requisit d'ordre doncs aquests si que s'executen ordenadament seguint els números de seqüència. Una vegada s'ha executat un *request* a una rèplica, aquesta envia el *reply* al client amb el resultat corresponent de la operació (r). Els altres paràmetres són com els del missatge de *request*: el timestamp t, el número de vista v, l'identificador del client c, l'identificador de la rèplica i i la seva signatura de clau pública σ_i.

[1] Són 2f i no 2f+1 missatges ja que el node primari no envia missatge de prepare

Els missatges de [REPLY,v,t,c,i,r] σ_i s'envien com es mostra a la figura 3.5.

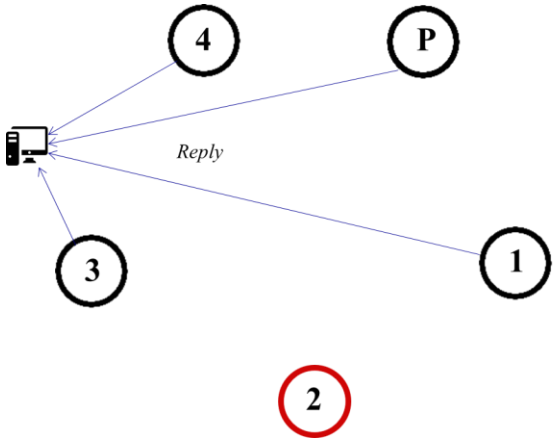

Figura 3.5 Missatge de reply del protocol PBFT

Cal recordar que el client espera f+1 replys provinents de diferents rèpliques abans d'acceptar el resultat r.

Es podria pensar que la fase de commit no és necessària doncs s'envia el missatge de pre-prepare i es fa l'acknowledge amb el missatge de prepare però no és així. És freqüent que hi hagi rèpliques que no rebin els suficient missatges de prepare degut a pèrdues de la xarxa però que més tard si rebin 2f+1 missatges de commit i per tant puguin "comitejar localment" un request. Inclús entre diferents vistes. Per tant, la fase de commit accelera el procés.

3.2.2 Mecanismes de recuperació de errors

Com s'ha mencionat anteriorment, els missatges de pre-prepare, prepare y commit s'emmagatzemen en els registres del nodes per provar que l'estat del sistema és el correcte. Al cap de moltes transaccions és un problema tenir ocupada tanta memòria i per tant, es necessita algun mecanisme per anar netejant aquests registres quan es fa *commit* d'una transacció. Aquest mecanisme s'anomena *checkpoint*.

Un *checkpoint* es genera cada x números de seqüència. On x variarà per cada sistema en concret. Si x és petit, s'emmagatzemen menys missatges a la vegada però s'envien més missatges de checkpoint. Per contra, si x és gran es guarden més missatges però s'envien menys missatges de checkpoint. El missatge de *checkpoint* té com arguments l'últim número de seqüencia *commitejat* correctament (n), el *digest* de l'estat (d) i l'identificador (i) i signatura (σ_i) de la rèplica que el crea.

Els missatges de [CHECKPOINT,n,d,i] σ_i s'envien com es mostra a la figura 3.6

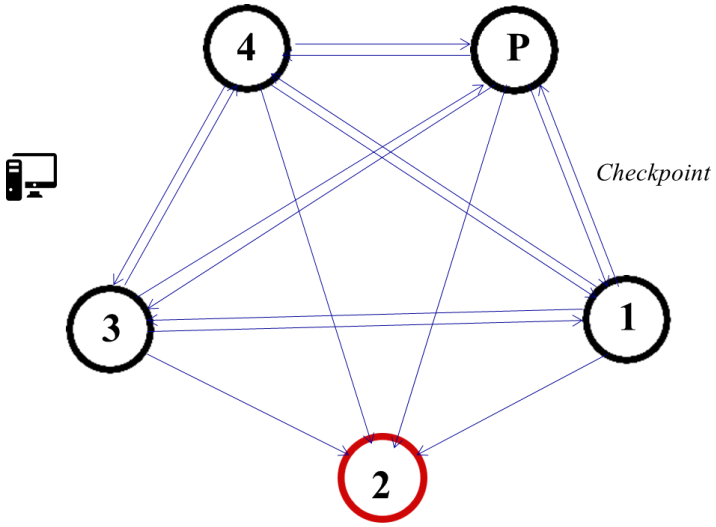

Figura 3.6 Missatge de checkpoint del protocol PBFT

Un checkpoint es diu que es estable quan una rèplica rep 2f+1 missatges de checkpoint incloent el seu. En aquest moment, la rèplica descarta tots els missatges guardats al seu registre amb el número de seqüencia menor o igual que n, incloent missatges de checkpoint anteriors. També s'augmenten els límits dels números de seqüencia h i H per donar cabuda a nous números de seqüència.

Apart d'aquest mecanisme també es necessita un de canvi de vista per garantir que l'algoritme continuí funcionant un cop es desconnecta el node primari o aquest ha sigut atacat maliciosament. Quan un client no rep cap reply al cap d'un determinat temps després d'enviar un request, fa un multicast d'aquest cap a les rèpliques *follower*. Aquests posen en marxa un temporitzador fins que rebin el pre-prepare del primari. Si el temporitzador caduca[2], la rèplica farà un multicast del missatge de canvi de vista o "new view". Aquest missatge inclou els següents paràmetres:

- v+1: número de la nova vista.
- N: número de seqüència de l'últim *checkpoint* estable s.
- C: conjunt de 2f+1 missatges de *checkpoint* que proven la validesa de s.
- P: conjunt de requests m amb número de seqüència > n. Si no es passessin en el missatge es perdrien, doncs no estan inclosos en el *checkpoint*.

[2] El que es coneix com a timeout

Els missatges de [VIEW-CHANGE,v+1,n,C,P,i]σ_i s'envien com es mostra a la figura 3.7.

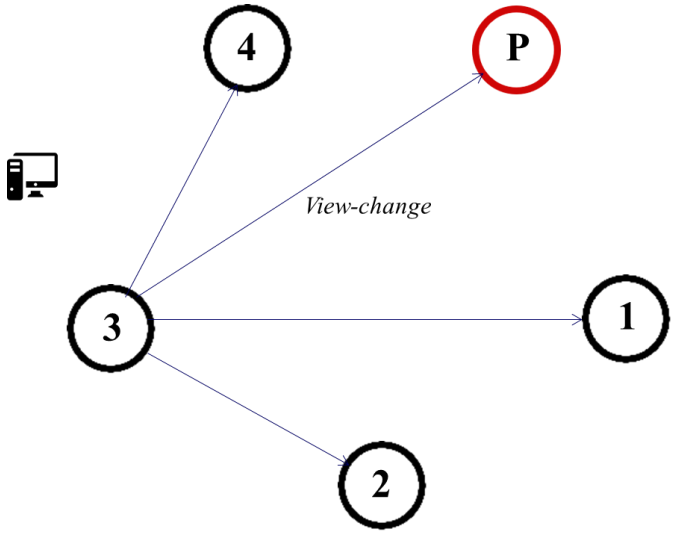

Figura 3.7 Missatge de view-change del protocol PBFT

Quan el primari de la nova vista (recordem que és aquell que compleix la fórmula P = v mod|R|) rep 2f missatges de canvi de vista, fa un multicast del missatge de nova vista que té els següents paràmetres:

- v + 1: número de la nova vista
- V: conjunt de 2f missatges de new-view rebuts pel primari.
- O: conjunt de missatges de pre-prepare de operacions no executades, construït a partir dels conjunts P de *requests* enviats pels altres nodes.

Els missatges de [NEW-VIEW,v+1,V,O]σ_p s'envien com es mostra a la figura 3.8.

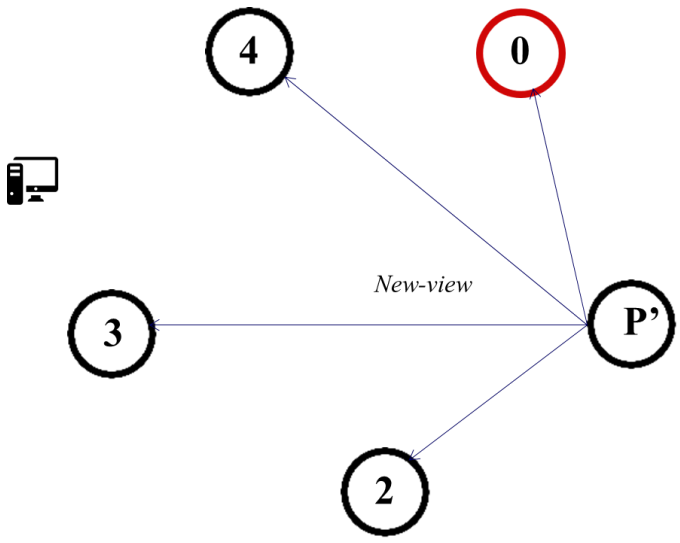

Figura 3.8 Missatge de new-view del protocol PBFT

Quan una rèplica rep el missatge de nova vista, entra a la vista v+1 i processa O. Mentre es fa el procés de canvi de vista, no s'accepten nous requests. Si després del procés a una rèplica li falta algun request o checkpoint estable, els podrà recuperar a partir d'una altra rèplica doncs aquesta té el conjunt C de missatges de certificació del chekpoint.

3.3. Valoració final de PBFT

Quan s'estudia un sistema basat en PBFT com el que hem vist, es pot tenir una primera impressió de que es podria desenvolupar en un entorn real, però no és així. L'inconvenient més gran que té aquest algoritme és la vulnerabilitat enfront dels atacs *sybil*. Aquest són els atacs que fa un individu o una organització introduint un número prou elevat de nodes al sistema com per controlar-lo. Doncs com s'ha vist, si 1/3 dels nodes són maliciosos, el sistema ja no pot funcionar.

Per evitar aquests atacs, els sistemes utilitzen el que s'anomena una llista de membresia, on s'indica quins nodes formen part de la xarxa. Això evita que no puguin entrar nous participant a la xarxa evitant els atacs *sybil* però no permet que sobre aquesta infraestructura es pugui donar cabuda a una criptomoneda doncs és necessari que sigui lliure l'entrada a la xarxa.

Un altre inconvenient és que el protocol no és escalable doncs s'envien molts missatges entre tots els nodes. Només cal observar que els missatges de prepare, commit, i checkpoint són multicast i per tant, en xarxes molt grans hi hauria molts retards. Tots aquests missatges porten signatures de clau pública i per tant un altre inconvenient és la manca d'anonimat en aquests sistemes.

No obstant això, PBFT crea unes bases que després han sigut utilitzades per crear infraestructures de moneda digital com veurem al capítol 5 amb el protocol Federated Byzantine Agreement que superarà aquestes limitacions.

4. Altres algoritmes per tractar errors bizantins

Hi ha moltes alternatives a PBFT per aconseguir consens en una blockchain on ocorren errors bizantins. Les més rellevants, utilitzades i les que analitzem a continuació són Proof-of-work(PoW) i Proof-of-stake(PoS). Entre d'altres més minoritàries podem trobar Proof-of-Capacity o delegated Proof-of-stake(PoS).

4.1. Proof-of-work

Aquest és de llarg el mecanisme més utilitzat per arribar al consens a les blockchain. És l'algoritme utilitzat per les gran conegudes Bitcoin, Ethereum i Litecoin, situades en el top 5 de les monedes digitals amb el valor de mercat més elevat.

En PoW no es necessari que tots els nodes de la xarxa arribin a una conclusió final sobre una transacció i que la comparteixin com és el cas de PBFT sinó que s'utilitzen funcions de hash per crear condicions, les quals seran suficients perquè un sol node imposi les seves conclusions sobre la transacció. Aquest procés s'anomena minat de bloc. Generalment aquest participant que aconsegueix minar el missatge rep un incentiu i d'aquesta manera s'impulsa als participants de la xarxa a minar. Aquestes conclusions del participant que ha minat el bloc podran ser verificades pels altres nodes del sistema però aquests ja no caldrà que les busquin per ells mateixos.

En molts moments és possible que la blockchain es bifurqui, doncs dos participants han minat un bloc, és probable que un d'ells sigui un node maliciós. Tanmateix si més del 50% dels nodes prenen per cert un camí, es decideix que aquest és el correcte. Tot i així, es pot produir el que s'anomena un atac del 51%, quan una persona o grup controla el 51% del poder computacional de la xarxa de manera que té més poder de minat que la resta i per tant, podrà controlar el sistema. En infraestructures tan grans com la de Bitcoin és gairebé impossible fer un atac d'aquest tipus ja que tenir més del 50% del poder computacional suposa un cost econòmic molt alt.

Aquesta tecnologia permet que qualsevol pugui participar a la xarxa de manera fàcil i de manera anònima i manté una estabilitat a la xarxa doncs els participants que minen, anomenats miners, volen que aquesta funcioni correctament ja que estan rebent incentius.

Els grans avantatges de PoW respecte PBFT són la descentralització de la xarxa doncs no existeix cap node que actuí com a primari en cap moment i l'anonimat dels

participants. La gran desavantatge és el gran cost elèctric que requereix el procés de minat de bloc.

4.2. Proof-of-stake

PoS és el segon mecanisme més utilitzat en aquest camp. Peercoin i Nxt són exemples de blockchain que utilitzen aquesta tecnologia. En PoS els miners s'anomenen validadors i la oportunitat de validar un bloc ja no dependrà de la força computacional que un tingui sinó en la quantitat de monedes que aquest disposi en el sistema. Per cada bloc a validar es fa un sorteig on la riquesa dins la xarxa és directament proporcional al percentatge de sortir com a escollit.

Les gran avantatges de PoS és l'estalvi d'energia respecte PoW en contra de la pèrdua de descentralització ja que els més rics són els que controlen el sistema. En aquests sistemes no es possible realitzar un atac del 51% ja que tenir molts nodes no implica controlar la xarxa. Apart, els participants més rics generalment estan interessats en el bon funcionament de la xarxa

4.3. Comparativa PBFT amb PoW i PoS

Una de les avantatges principals de PBFT respecte les altres és la capacitat per fer *commit* de les transaccions sense la necessitat de confirmacions amb el procés de mineria o de validació, donant al sistema temps d'execució més baixos. Per exemple en el cas de Bitcoin, el temps de minat de bloc sol durar 10 minuts mentre que en PBFT sol durar segons. Respecte PoW una gran avantatge de PBFT és la gran reducció dels costos energètics al no realitzar cap operació matemàticament complexa. Respecte PoS una gran avantatge és la descentralització que ofereix el protocol doncs tot i tenir la figura de node principal, aquesta va rotant i no té un pes molt important en el procés de *commit* d'una transacció. Tot i així, els sistemes PBFT no arriben a la descentralització total dels sistemes PoW.

5. Federated Byzantine Agreement

S'ha vist com el protocol PBFT té grans punts forts respecte els seus principals competidors, PoW i PoS però té dues limitacions massa importants com per funcionar per ella mateixa i és per això que neix la tecnologia Federated Byzantine Agreement, coneguda amb les sigles FBA i desenvolupada per l'empresa Stellar.

FBA inclou les següents millores respecte PBFT:

- Confiança flexible
- Seguretat asimptòtica
- Baixa latència

5.1 Confiança flexible

Al protocol PBFT els nodes confien en el sistema en si, si tot el sistema compleix la premissa de n> 3f + 1, en general els nodes podran fer *commit* dels missatges. Però si no es compleix, serà impossible realitzar-ho i per això el protocol és molt vulnerable a atacs. Al protocol FBA no serà necessari complir amb aquesta premissa sinó que cada node podrà escollir de quins altres nodes es fia formant el que s'anomena un *quòrum slice.*

Un node podrà tenir més d'un *quòrum slice* format per diferents nodes i només que un d'ells compleixi la premissa de n' > 3f + 1 serà suficient per comitejar una transacció. On n' serà el número de nodes del *quòrum slice*. A la figura 5.1 es pot observar com el node 1 té una *slice* amb dos nodes maliciosos però això no li frenarà en el procés de fer commit d'una transacció perquè té una altre *slice* que si li permet.

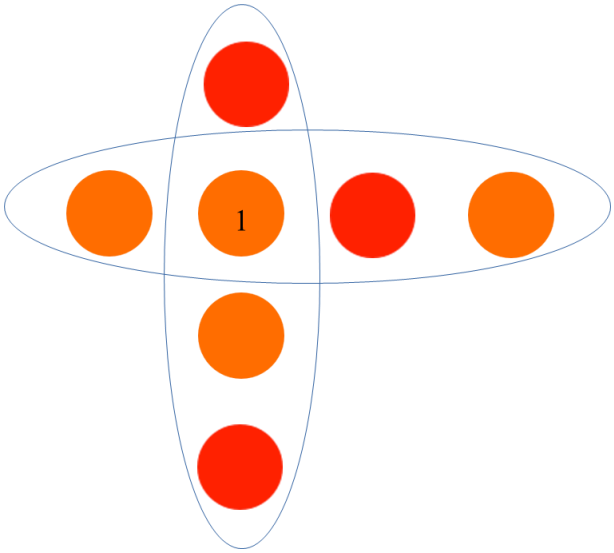

Figura 5.1 Quòrum slices del node 1 del protocol FBA. Els nodes taronges representen nodes lleials mentre els vermells representen nodes maliciosos

Per poder garantir que s'arribarà a un acord entre els nodes utilitzant quòrum slices s'ha de complir amb la condició de intersecció dels quòrums. És a dir, els quòrum slices s'han de solapar com es veu a la figura 5.2. En el cas de que aquests quòrums quedin disjunts com es pot observar a la figura 5.3, no es podrà garantir el correcte funcionament de la blockchain, ja que els missatges de verificació no podran enviar-se entre slices. És molt important que cada sistema que utilitzi FBA utilitzi un algoritme de formació dels *quòrum slices* que garanteixi aquesta intersecció.

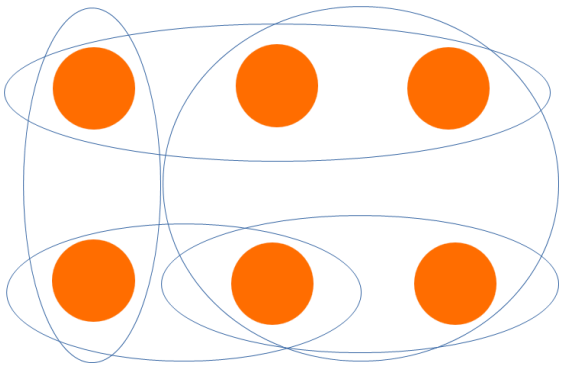

Figura 5.2 Quòrum slices complint intersecció

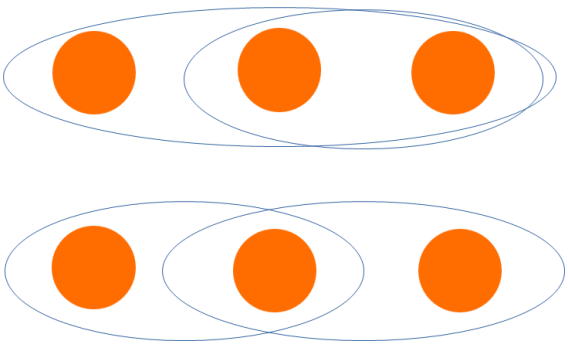

Figura 5.3 Quòrum slices disjunts

5.2 Seguretat asimptòtica

Amb la tecnologia de les *quòrum slices* no només permet una confiança flexible sinó que permet una defensa contra els atacs del 51% doncs encara que un atacant controli la majoria del sistema, els nodes lleials confiaran entre ells i no pas en els nodes maliciosos que acaben d'entrar al sistema.

En conseqüència, ja no és necessària una llista de membresia doncs qualsevol node pot entrar al sistema sense perjudicar-lo i així es supera una de les limitacions més importants que tenia PBFT. Apart el fet de que cada node pugui escollir de qui confia implica una descentralització més elevada.

Per concloure amb el protocol FBA, és necessari dir que no millora la desavantatge de la manca d'anonimat però si que s'ha sobreposat a les altres limitacions.

5.3 Baixa latència

La tecnologia de les *quòrum slices* permet que no s'hagin d'enviar tots els missatges entre absolutament tots els nodes de la xarxa sinó que tots els missatges de multicast vistos a l'algoritme PBFT només s'enviaran els nodes que pertanyin a la mateixa *quòrum slice*. Aquest fet produeix una baixada de la latència.

6. Blockchain basades en PBFT i valor de la criptomoneda al mercat

A continuació es descriu algunes de les criptomonedes amb més pes actualment així com quin tipus de protocol utilitzen i el seu corresponent valor al mercat.

6.2. Stellar

Stellar és la primera blockchain que surt al mercat amb tecnologia PBFT, al ser ells els creadors del protocol Federated Byzantine Agreement a l'any 2013. La seva moneda s'anomena XLM. A partir del moment en el que va sortir a la llum aquesta tecnologia, altres empreses van començar a investigar i desenvolupar altres aplicacions utilitzant la tecnologia FBA.

Capitalització del mercat[3]: 1.838 Milions de $
Volum en les últimes 24 hores: 204 Milions de $

6.2 Cosmos

Cosmos és la blockchain de l'empresa Tendermint creada al 2016. La seva moneda s'anomena ATOM. Cosmos utilitza un protocol BFT Proof-of-Stake, és a dir, junta algunes de les propietats pròpies de la família de PBFT amb d'altres pròpies de PoS.

Recordem que al protocol PBFT hi ha una rèplica que actua de líder, habitualment escollida a partir del seu identificador, és a dir, un mètode bastant aleatori on totes les rèpliques tenen la mateixa probabilitat. Així doncs, una replica maliciosa té les mateixes possibilitats d'agafar aquest rol de líder que el node amb més riquesa de la xarxa. En Cosmos, aquesta igualtat no existeix ja que per cada bloc a validar el líder serà escollit amb el model de PoS, així doncs, com més riquesa tingui un node més probable serà que surti escollit.

Capitalització del mercat: 866 Milions de $
Volum en les últimes 24 hores: 64 Milions de $

[3] Valors del dia 6 de Maig de 2019 a les 12.23 h

6.3. Zilliqa

Zilliqa és una blockchain molt nova, va sortir a la llum al gener del 2018. La seva moneda s'anomena ZIL. Els creadors d'aquesta moneda tenien un objectiu molt clar, competir contra les altres blockchain amb un temps de transacció molt baix, exactament aconseguir 2400 transaccions per segon.

Utilitzen tecnologies PBFT per aconseguir aquesta velocitat. Recordem que en PoW els temps de bloc són molt més alts, per exemple en Bitcoin sol ser de 10 minuts. Apart també aconsegueixen que en el moment d'augmentar el número de processos, la xarxa sigui funcionant correctament amb la tecnologia FBA.

Capitalització del mercat: 134 Milions de $
Volum en les últimes 24 hores: 13,8 Milions de $

7. Conclusions

En aquest document s'han vist diferents maneres d'enfocar el problema del consens a una *blockchain*. En primer lloc una visió d'un model on no pot ocórrer cap tipus d'atac cap al nostre sistema. Tot seguit s'han introduït aquests atacs a l'hora de plantejar i desenvolupar els algoritmes de consens, específicament el protocol *Practical Byzantine Fault Tolerance*.

S'ha vist les limitacions que té aquest algoritme a l'hora d'aplicar-se a un entorn real, així com una comparació amb altres tipus de protocols que també s'utilitzen per tractar els errors bizantins. Per tant, de primeres es pot dir que la nostra hipòtesi inicial no era encertada.

S'ha explicat el protocol *Federated Byzantine Agreement*, que utilitza les bases de PBFT i que aconsegueix que sistemes que utilitzin aquesta tecnologia puguin tenir cabuda a un entorn real. FBA no millora la desavantatge que té PBFT de la manca de anonimat però si comparem amb PoW, si que resulta ser un protocol amb un cost energètic molt menys elevat. Aquest és un punt a favor quan es tenen en compte les implicacions ambientals. Per tant, tornant a la nostra hipòtesi inicial, és veritat que no es pot substituir PoW per PBFT però si per tecnologies que deriven directament d'aquesta.

És important mencionar que aquests protocols s'hauran d'anar actualitzant doncs és molt possible que en els pròxims anys es descobreixin noves eines més potents a l'hora d'atacar els sistemes. Aquesta pot ser una possible continuació del nostre treball, estudiar i analitzar nous protocols que es desenvolupin a partir de PBFT i FBA.

Per acabar, s'ha assolit l'objectiu de generar material docent en forma de diapositives dels protocols *Zookeeper Atomic Broadcast, Practical Byzantine Fault Tolerance* i *Federated Byzantine Agreement.* Així com una simulació en python de l'algoritme *Zookeeper Atomic Broadcast* per observar diferents comportaments del sistema depenent de l'estat dels nodes que el formen. Aquest material ha sigut adjuntat en forma d'annex en el dipòsit del TFG.

Bibliografia

Per la realització d'aquest projecte s'han consultat habitualment els *papers* dels protocols estudiats així com alguna publicació d'autors independents.

[BT1985] G. Bracha, S. Toueg "Asynchronous Consensus and Broadcast Protocols". Journal of the Association for Computing Machinery, Vol. 32, No. 4, Octubre 1985, pàgines 824-840.

[Cas2001] M. Castro "Practical Byzantine Fault Tolerance". MIT Laboratory for Computer Science, 2001. [Online] Disponible: http://pmg.csail.mit.edu/~castro/thesis.pdf [Accedit: 12 de novembre de 2018]

[CL1999] M. Castro, B. Liskov. "Practical Byzantine Fault Tolerance". MIT Laboratory for Computer Science, 1999. [Online] Disponible: http://pmg.csail.mit.edu/papers/osdi99.pdf [Accedit: 10 de novembre de 2018]

[CL2002] M. Castro, B. Liskov "ACM Transactions on Computer Systems". MIT Laboratory for Computer Science, vol. 20, no. 4, pàgines 398–461, novembre 2002.

[Dou2002] J. Douceur "The Sybil Attack". Microsoft Research. [Online] Disponible: https://www.freehaven.net/anonbib/cache/sybil.pdf [Accedit: 15 d'octubre de 2018]

[Lam1983] L. Lamport "The Weak Byzantine Generals Problem". Journal of the Association for Computing Machinery, vol 30, No 3, pàgines 668-676, Juliol 1983.

[LSP1982] L. Lamport, R. Shostak, M.Pease "The Byzantine Generals Problem". ACM Transactions on Programming Languages and Systems, vol. 4, No. 3, pàgines 382-401, juliol 1982.

[Maz2016] D. MAZIERES. "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus". Stellar Development Foundation, 2016. [Online] Disponible: https://www.stellar.org/papers/stellar-consensus-protocol.pdf [Accedit: 10 febrer de 2019]

[Med2012] A. Medeiros "ZooKeeper's atomic broadcast protocol: Theory and practice". 2012. [Online] Disponible: http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf. [Accedit: 10 de novembre de 2018]

[Nak2008] S. Nakamoto "Bitcoin: A peer-to-peer electronic cash system" 2008. [Online] Disponible: https://bitcoin.org/bitcoin.pdf [Accedit: 8 d'octubre de 2018]

[RJ2008] B. Reed, F.P. Junqueira. "A simple totally ordered broadcast protocol" Yahoo! Research, 2008. [Online] Disponible: https://www.datadoghq.com/pdf/zab.totally-ordered-broadcast-protocol.2008.pdf. Accedit: 4 de novembre de 2018]

Top 100 Cryptocurrencies by Market Capitalization. [Online] Disponible: https://coinmarketcap.com/ [Accedit: 6 de maig de 2019]